An Intelligent Message System

Implemented as a Data Object

by

Cynthia L. Humphrey

B. S. Millersville University, 1979

A Master's Report

submitted in partial fulfillment of the

requirements for the degree

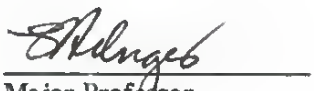
Master of Science

Department of Computer Science

Kansas State University
Manhattan, Kansas

1987

Approved by:



Major Professor

CONTENTS

1. CHAPTER ONE - Introduction..... 1
1.1 Overview..... 1
1.2 General Ideas and Terms..... 4
1.2.1 Office Information Systems..... 4
1.2.2 Intelligent Messages..... 4
1.2.3 Data Objects..... 5

2. CHAPTER TWO - Comparison of Systems..... 8
2.1 Form Management Systems..... 8
2.1.1 Officetalk-Zero and Officetalk-D..... 9
2.1.2 SCOOP..... 9
2.1.3 Odyssey..... 10
2.1.4 OFS..... 10
2.2 Message Systems..... 11
2.2.1 Message Management Systems..... 11
2.2.2 Intelligent Message Systems..... 12
2.2.2.1 R2D2..... 12
2.2.2.2 IMAIL..... 13
2.2.2.3 XMAIL..... 13

3. CHAPTER THREE - Requirements..... 16
3.1 General Requirements..... 16
3.2 System Requirements..... 16
3.2.1 SENDER - INITIATE Process..... 18
3.2.2 RECEIVER Process..... 20
3.2.3 SENDER - RECEIVE RESPONSE Process..... 21

4. CHAPTER FOUR - Design..... 23
4.1 Design Assumptions..... 23
4.2 System Invocation and Process Interaction..... 24
4.3 Module Description and File Description..... 28
4.3.1 General Description..... 28
4.3.2 Modules and Files Necessary by Process..... 30
4.4 Data Object Design..... 31
4.5 Directory Structure, File Location, and File Naming..... 33
4.6 Mail Program Interaction..... 35

5. CHAPTER FIVE - Implementation..... 36

6. CHAPTER SIX - Conclusions and Extensions..... 38

APPENDIX 1 - Page 1..... 40

APPENDIX 1 - Page 2..... 41
APPENDIX 2 - Page 1..... 42
APPENDIX 2 - Page 2..... 43
APPENDIX 2 - Page 3..... 45
APPENDIX 2 - Page 4..... 46
APPENDIX 2 - Page 5..... 47
APPENDIX 2 - Page 6..... 48
APPENDIX 2 - Page 7..... 49
APPENDIX 2 - Page 8..... 50
APPENDIX 2 - Page 9..... 51
APPENDIX 2 - Page 10..... 52
APPENDIX 2 - Page 11..... 53
APPENDIX 2 - Page 12..... 56
APPENDIX 3 - Page 1..... 58
APPENDIX 3 - Page 2..... 59
APPENDIX 3 - Page 3..... 61
APPENDIX 3 - Page 4..... 63
APPENDIX 3 - Page 5..... 64
BIBLIOGRAPHY..... 66

LIST OF FIGURES

Figure 1. SENDER - INITIATE..... 18

Figure 2. RECEIVER..... 20

Figure 3. SENDER - RECEIVE RESPONSE..... 21

Figure 4. PROCESS INTERACTION..... 25

Figure 5. DATA OBJECT STRUCTURE..... 31

1. *CHAPTER ONE - Introduction*

1.1 *Overview*

This paper introduces a system which implements intelligent messages, also called active messages. Intelligent messages are executable objects with the ability to perform some actions and interact with the receiver [VITT81].

In addition to acting as an intelligent message, the message acts as a data object that contains routing information and both data and program code [COHE84] to retrieve, manipulate and store the data. This is in essence an Intelligent Data Object - a self-sufficient capsule of information that can be routed through a network.

'XMAIL' utilizes an existing mail facility to interactively poll users for questionnaire-type information. It retrieves a user's current information from the data base, creates the intelligent data object(IDO), mails the IDO, receives and executes the object, returns the object, updates the data base with the information, and performs various status checks. The system was implemented to replace a paper questionnaire on employee status, e.g., address, sex,

marital status, degree and school name and address, in-house courses taken, etc. The questionnaire content can be tailored to suit the user needs, since the questionnaire is the application to be used with 'XMAIL'. 'XMAIL' was implemented on a VAX 11/780 under the 4.2BSD (Berkeley Software Distribution) UNIX* Operating System, using the 'C' language and Shell commands. The Shell is a command interpreter for the UNIX operating system. The data base management system used is INGRES** , a relational data base system currently available on UNIX.

This paper is organized in six chapters. The introduction continues with a discussion of office information systems, intelligent messages and data objects. The second chapter contains a review of the current systems related to intelligent messages and intelligent data objects, and how these systems relate to 'XMAIL'. The third chapter describes the requirements for the system - what is necessary for the system to be useful. The fourth chapter includes

---

* UNIX is a registered trademark of AT&T Bell Laboratories.

** INGRES is a registered trademark of Relational Technology, Inc.

detailed design information, including data structures and diagrams. The fifth chapter describes how the system was implemented and tested. The final chapter is a conclusion of the problems encountered, an analysis of the final system and some possible future enhancements.

1.2 *General Ideas and Terms*

1.2.1 *Office Information Systems* Ellis and Nutt define office automation systems as "entities which perform document storage, retrieval, manipulation, and control within a distributed environment." The advantage of an office information system is that it increases the efficiency of the office through the controlling of the flow of information and of the user's interfaces to the system. For example, one user interface could control the creation, sending, receiving and filing of a form [ELLI80].

Office information systems can be very varied. One possible use is in the mail system environment. Communication among office personnel has traditionally been done by paper messages, and later by electronic but passive mail messages - ones that are created, sent, and received. The messages merely relayed information. Any responses would have to be initiated by the receiver [HOGG84].

1.2.2 *Intelligent Messages* Intelligent messages, also called active messages, are executable objects with the ability to perform some actions and interact with the receiver [VITT81]. They return the responses received to the original sender, and can route themselves to

others.

In the basic questionnaire, questions are asked and answers given. If it is a paper questionnaire, the answers are usually on the same sheet of paper as the questions. In the traditional electronic message system, the receiver must give his/her answers in a return message. Once all answers are received or a cutoff date has been reached, the answers are tallied. Tallying the answers usually requires some human intervention [VITT81].

An intelligent message in questionnaire format could ask a question, get a response and determine the next question based on this response. In this way, questionnaire branching is transparent to the person responding. The responses could be sent back to the originator and automatically stored and tallied [VITT81].

1.2.3 *Data Objects* In object oriented programming, data and the programs that access it are contained in one program unit called an object. The details of the object are encapsulized (localized and concealed) to make it easier for the programmer to understand and use. Abstraction also localizes and conceals the

pattern of generation and use of the object. An abstract data type is a class of abstract objects defined by the set of operations performed on them [COHE84, LISK74, GRIE77]. (User defined types can be looked at as a pattern for a data structure).

An intelligent message can be thought of as an object. It can contain routing information, data, and the code to retrieve, manipulate and store the data. This is in essence an Intelligent Data Object(IDO) - a self-sufficient capsule of information that can be routed in a network. Each intelligent message copy is an instance of the object. The message is activated sometime after arriving at its destination. It is actually self-modifying since the data can be changed.

Data objects are not new to message processing. In a slightly different form, in languages like LISP programs can create other programs and execute them; data is program and executable. Some languages are purely object-oriented and contain only objects, such as Smalltalk80. The data type is encapsulated inside a set of procedures which know how to manipulate that data type [COX84].

The next chapter reviews the current systems related to intelligent messages and intelligent data objects.

2. *CHAPTER TWO - Comparison of Systems*

This chapter contains a review of the current systems related to intelligent messages and intelligent data objects, and how these systems relate to 'XMAIL'.

2.1 *Form Management Systems*

One type of office communication is the form. The form is an integral part of the information gathering process in business today. It is the central structure for data flow in an office. Common business forms such as invoices and checks are used for the collection, storage, retrieval, and update of data. An electronic form is intended to be a logical representation of a physical, or paper, form. The form system supports the insertion, update, deletion and retrieval of form instances from a logical file. A form instance is defined to be a completed form, where the fields have values. A logical file can be thought of as the logical version of a file cabinet drawer that holds folders of forms [SORE79]. An intelligent form requires information fields, a routing list, instructions for the recipient, and a check-list of people who are to process the form [MCBR83].

There are several office information systems which are

form management systems. A few of these systems are described in the remainder of this section. These were selected to describe the sources of the concepts which contributed to this system.

2.1.1 *Officetalk-Zero and Officetalk-D* Officetalk-Zero is an office information system that is based upon the data objects of single page forms and files of forms to be passed among the office workers. It is intended to aid in document management, preparation and communication. Some of the subsystems include a text editor, graphics package, communication facility, filing facility, and forms data entry capability. The mail system involved does send and receive operations and is not intelligent mail [ELLI80]. Officetalk-D is an extension of Officetalk-Zero. It was designed to support a distributed office information system and to make the utilization of the data base transparent to the user [BERN82].

2.1.2 *SCOOP* Another office information system is SCOOP(System for Computerization of Office Procedures). SCOOP's emphasis is on automating office procedures or processes rather than individual tasks. Its subsystems include document generators, mail senders and receivers, file services and media schedulers. SCOOP also does not have an intelligent mail system [ZISM78].

2.1.3 *Odyssey* The office information system Odyssey is a business-trip planning form system. It is also a knowledge-based assistance system, where the data objects react to new data sent to them, supply default data when needed, and compute or seek missing data when it is requested. In the execution of these functions the form interacts with the data base. There was no mention of a mail capability [FIKE81].

2.1.4 *OFS* OFS(Office Form System) involves the automating of individual desk activities as well as the coordination of activities between work stations. The mail activity of OFS involves the automatic routing of mail, but no intelligent mail [TSIC80].

Several other form management systems exist, such as FORMANAGER [YAO84] and Officetalk-D [ELLI82], but do not have intelligent mail capabilities.

2.2 *Message Systems*

Mail is the chief written communication medium today. Electronic mail is highly important in the automated office since it provides the transfer mechanism for the moving of information related to office activities [STEF81]. In order to help make the office more efficient and productive, many message systems have been developed.

2.2.1 *Message Management Systems* One type of message system is the message management system. The prototype system of Tsichritzis, Rabitti, Gibbs, Nierstrasz, and Hogg is a message management system which utilizes a data base management system to 'manage' the messages. This system 'manages' the messages according to their content. The messages reside in the data base. The messages are structurally typed, that is, structurally consistent for each message type. They are considered located at one node but can be sent from node to node. The system can continuously look for certain messages, can query on existing messages, and can perform actions based on messages [TSIC82]. In essence, this is not intelligent mail.

Message management systems are also discussed by Mazer and Lochovsky [MAZE83].

2.2.2 *Intelligent Message Systems* Intelligent message systems are those which incorporate intelligence into the message itself, such as the ability to interact with the user and receive responses.

An intelligent questionnaire contains:

1. A language to specify the structure and content of the questionnaire

2. A mechanism to send these instructions in a message and have them evaluated by the message system

3. A mechanism for executing these instructions when the intelligent questionnaire is invoked

4. A mechanism to send replies back to the originator [VITT81].

In the 'XMAIL' system an interface with a data base is also a requirement.

2.2.2.1 *R2D2* One such intelligent message system, R2D2(Research-to-Development-Tool for Message Processing), contains instructions in its own language to tailor the message [VITT81]. The message is only textual in nature. The return message contains the original message and the responses, also textual in

nature. Automatic routing of the questionnaire was implemented.

2.2.2.2 *IMAIL* IMAIL is also a system that uses instructions in its own language. Its interfaces are similar to **UNIX** Mail [UNIX84]. The responses are contained in local and global variables, global variables existing from invocation to invocation of the intelligent message. In essence the intelligent message and its copies exist until they are removed with a terminate command [HOGG84].

2.2.2.3 *XMAIL* 'XMAIL' was created with the intent of being implemented through the **UNIX** Mail system as an intelligent message system. Its use is practically transparent to the user. The system administrator invokes the system and the user views the results as a 'live' message. 'XMAIL' was created with the questionnaire/polling system objective even though the data could be considered in form style. The message is not routed among users since it is really a personalized message which includes the user's current data base information. The primary advantage of this system is its suitability for remote polling when the results are to be installed in a local data base. 'XMAIL' also could be used for low-use periodic automatic updates. Since 'XMAIL' is compatible with a

data base management system, information about a form can be obtained through 'XMAIL', where processing is one at a time, or through the data base management system facilities, where the specific queries are available to operate on the collection of forms. This allows information about a number of forms to be obtained and processed.

'XMAIL' not only accepts new responses but contains the existing responses that were extracted from the data base. The high-level language executable code included as part of the message interacts with the user and modifies this information. Because of the use of a high-level language, the message content can be quite complex.

'XMAIL' could be considered a form system since it could be used to update the fields of a form. However, it is specific in that it only implements the form or questionnaire specified in the executable code supplied by the administrator - it is not a user-defined form system. The code supplied by the administrator interacts with the receiver, gets responses, and stores the responses in a data file. See Section 4.3 for a description of the modules and files from which this system is composed. 'XMAIL' could be modified to route

the message among users for completion of a particular form; however, this was not the intention of this system. These functions could be introduced at a later date, or capabilities existing elsewhere could be combined with this system.

The next chapter describes the requirements for the 'XMAIL' system - what the system must do in order to be useful or successful.

3. *CHAPTER THREE - Requirements*

This chapter defines the requirements for the 'XMAIL' system - what the system must do to be successful. The system is broken down logically into three processes. Each of these processes is described in terms of the functions they perform.

3.1 *General Requirements*

'XMAIL' must be able to perform the following general functions:

- create a data object
- send a data object
- receive and execute a data object
- return a data object
- retrieve and update information from a data base
- allow the questionnaire/form content and data format to be supplied by the administrator

3.2 *System Requirements*

'XMAIL' is an intelligent message system that performs many separate functions. It produces the desired results of creating a data object, sending the data

object, receiving and executing the data object, and returning the data object, in conjunction with the use of a data base to retrieve and store the data.

'XMAIL' could be divided into three logical processes that include the above functions. They are:

1. sender - initiate
2. receiver
3. sender - receive response

The following sections describe each of the processes in terms of the functions they perform. The figures included represent Structured Analysis (SA) Data Diagrams. The boxes represent objects or data and the arrows represent activities. The figures show the activity flow of the system.

3.2.1 *SENDER - INITIATE Process*

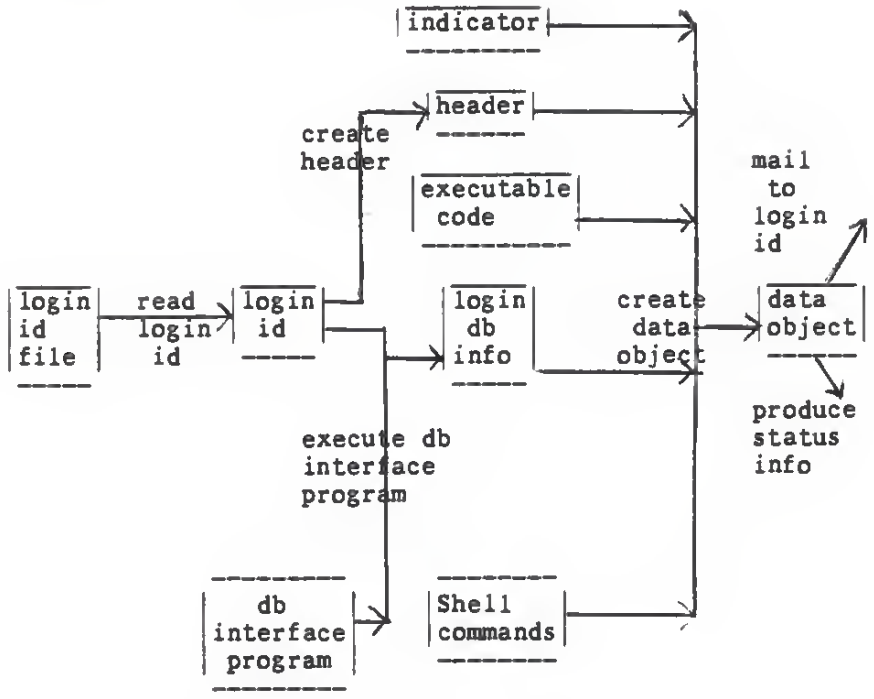


Figure 1. SENDER - INITIATE

The "sender - initiate" process performs the following functions:

- obtain a receiver's login id
- initiate a request to the data base for the retrieval of the receiver's current information
- retrieve the receiver's current data base information

- create a data object containing a data object indicator, header information, executable code, the receiver's current data base information, and a set of Shell commands
- mail the data object to the receiver
- produce status information indicating mail was sent

3.2.2 *RECEIVER Process*

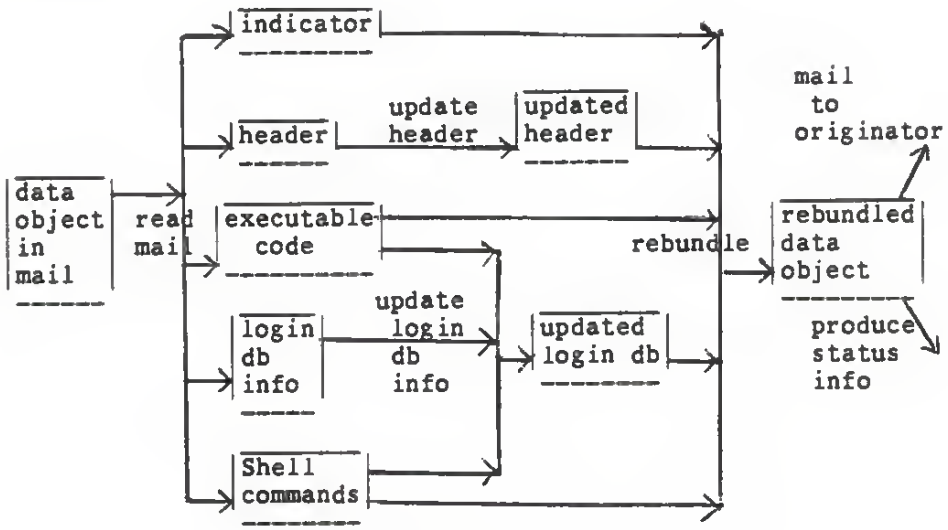


Figure 2. RECEIVER

The "receiver" process performs the following functions:

- recognize the data object, unbundle the pieces of the data object, and execute the set of Shell commands
- execute the executable code which will receive the responses and store them in the data area
- produce status information indicating mail was received
- rebundle the pieces of the data object and mail it back to the originator

3.2.3 *SENDER - RECEIVE RESPONSE Process*

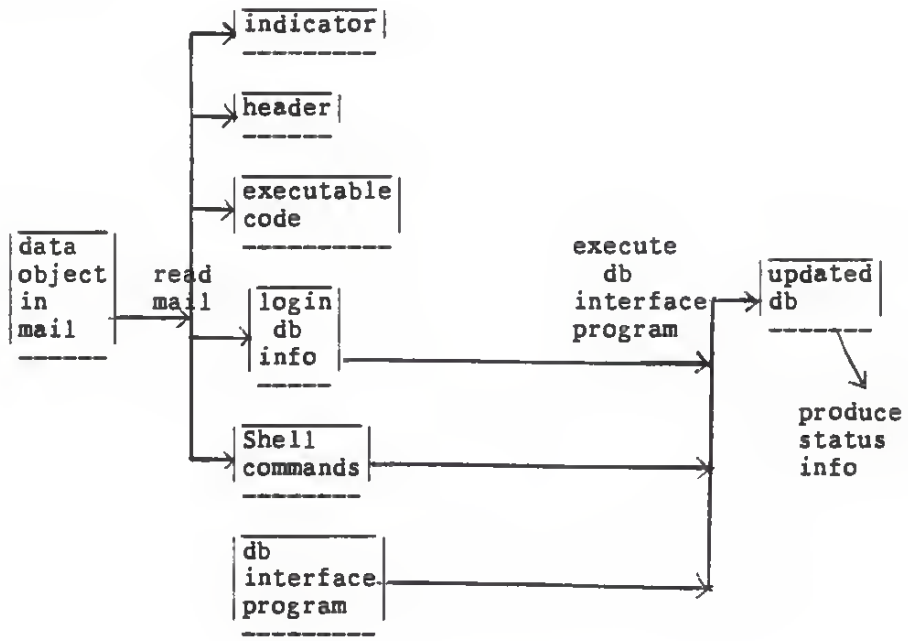


Figure 3. SENDER - RECEIVE RESPONSE

The "sender - receive response" process performs the following functions:

- recognize the data object, unbundle the pieces of the data object, and execute the set of Shell commands
- execute a local data base interface program which will initiate an update of the data base using the newly updated information

- produce status information indicating the data base was updated

The next chapter describes the design of 'XMAIL', including design assumptions and how the system was designed to meet the requirements.

4. *CHAPTER FOUR - Design*

This chapter describes the design of 'XMAIL', including the design assumptions, system invocation and process interaction, module description and file description, data object design, directory structure, and Mail program interaction.

4.1 *Design Assumptions*

There were several assumptions made during the design of this system. The assumptions were:

1. The system will be implemented using a centralized data base.

2. It will be possible to modify the existing system Mail command.

3. Since remote execution of programs can be a security threat, the system will be implemented on a local network.

4. The system will be implemented on a VAX 11/780 under the 4.2BSD UNIX environment, using the 'C' language and Shell commands.

5. The data base management system utilized will be INGRES, although the system can be modified for other data base systems.

6. All persons having data in the data base will be assumed to have a mail login id in the network.

4.2 *System Invocation and Process Interaction*

This system was designed for ease of use, by allowing the entire system process ('sender - initiate', 'receiver', and 'sender - receive response' processes) for the entire user community to be done by the "push of a button". Once the system is activated the entire process will be automatic. Since there are status checks done throughout the cycle of the system operation, any operational problems, e.g. the response being "lost in the mail", will be recognized. For example, if a response has not been received after a fixed period of time, the system administrator will initiate a resending of the data object.

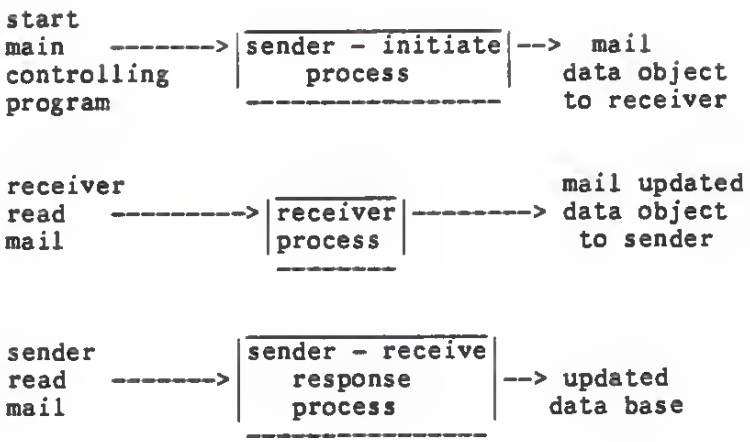


Figure 4. PROCESS INTERACTION

The process interaction of 'XMAIL' is illustrated in Figure 4. To start the system, the administrator will execute the main sending program. The main sending program is executed by typing:

XSEND [login_id]

If a login_id is specified, the system will send an intelligent message to one user. If a login_id is not specified, the system will send intelligent messages to several users. The users are specified in a login id file supplied by the administrator. This file can be modified to change the recipients of the message. See section 4.5 for a description of the directory structure, file location, and file naming of all of the

programs and files in the system. See Appendix 1 for a manual page on this command. The 'XSEND' program will start the process described in section 3.1(System Requirements) as the "sender - initiate" process. This process will be completed for every user in the login id file. The end result of this process is that the intelligent data object is mailed to the receiver. Multiple processes can exist at this point since each user is in control of a separate system process. The system is in effect halted for this receiver until the receiver initiates a reading of his/her mail and executes the message. See Appendix 1 for a manual page on the changes to the 'Mail' command. This act will start the "receiver" process. The end result of this process is that the updated data object is mailed back to the sender. The system is again halted for this receiver until the sender initiates a reading of his/her mail and executes the message. This act will start the "sender - receive response" process which will update the data base. The entire process is then complete for a certain user.

This system performs certain functions in the sender's environment and the receiver's environment. This is because the intelligent data object is created by the system in the sender's environment and is sent to the

receiver and executed there. It is also because an important part of 'XMAIL' is the modification of the Mail command. The Mail command initiates the execution of the data object in the receiver's environment and also initiates the update of the data base in the sender's environment once the response is received. Although it is logically one intelligent data object, the 'XMAIL' message consists of several programs and files, not all of which reside in the data object.

4.3 *Module Description and File Description*

4.3.1 *General Description* 'XMAIL' consists of several data files, "C" programs, and Shell programs. The units of the system required are:

1. login id file consisting of the list of users to receive intelligent messages (optional)
2. main sending program
    a. get header information (e.g. sender's login id, receiver's id, status flag)
    b. initiate data base interface program
    c. create data object
    d. mail data object to receiver
    e. print status information
3. data base interface programs
    a. retrieve receiver's current data base information and put in a file
    b. store receiver's updated data base information from a file
4. modified Mail program

a. recognize a data object

b. unbundle the data object into pieces

c. execute Shell controlling program

d. rebundle the data object

e. mail the data object back to the sender

f. clean up files (remove subdirectory created)

5. executable code

a. interact with the receiver and get responses

b. store responses in the data file

6. Shell controlling program in the data object

a. mail status information to the sender

b. execute the executable code

c. mail updated data object to sender

d. initiate the data base interface program

e. print status information

The data base interface programs and executable code (application code) will not be described in detail in this paper since they were implemented as a separate project. Their design is basically transparent to the system code, except that the program and data file names must be standard.

4.3.2 *Modules and Files Necessary by Process* The modules and files necessary by process are as follows:

SENDER-INITIATE	RECEIVER	SENDER-RECEIVE RESPONSE
login id file	access to modified Mail program	access to modified Mail program
access to main sending program		db update program
access to modified Mail program		
db retrieval program		
executable code		
copy of the Shell controlling program		

4.4 *Data Object Design*

The data object consists of five parts - the data object indicator, header, executable code, data, and Shell commands.

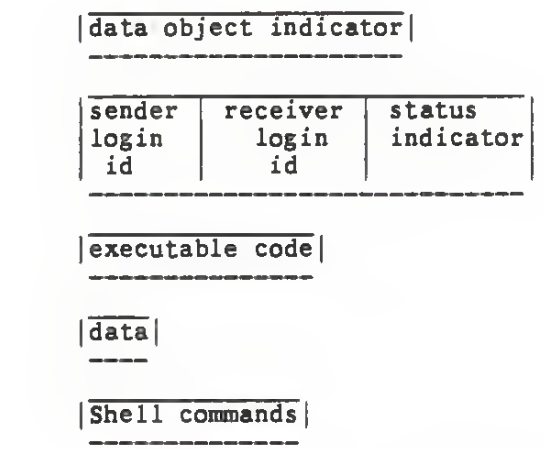


Figure 5. DATA OBJECT STRUCTURE

A data object indicator pattern is one such as '*** DATA OBJECT***'. The header consists of the sender's login id, the receiver's login id, and a data object status indicator, used for status checking. The possible values for the status indicator are:

- 1 - data object sent
- 2 - data object received

- 3 - data object response mailed

The executable code is written in "C". The code could be tailored to any need as long as it and the data base interface program access the data in the same manner. The data is created by the data base interface program prior to the creation of the data object, and is modified during the execution of the data object. The Shell commands do not need to be modified if the executable code and data base interface program change.

4.5 *Directory Structure, File Location, and File Naming*

The modified Mail program will be installed as the regular system Mail command. The main sending program, called 'XSEND', will also be installed as a system command. There are four fixed file names, three required because of the required knowledge of the names in the bundling process. They are:

```
data base data file ---------- 'im_data'
executable file -------------- 'im_code'
Shell controlling program ----- 'shell1'
login id file ----------------- 'user.file'
```

The names of the data base retreival and update programs may be names of your choice since they are prompted for. The data base retreival program may be located anywhere as long as it places the created data base data file in the current directory under the name 'im_data'. The data base update program may be located anywhere as long as it knows where to access the updated data file deposited by the 'XMAIL' system (user is prompted for where to deposit the updated data file). The data base update program must also know to access the 'im_data' as '${receiver}im_data'. This renaming when the file is deposited is done to ensure that the file will not be lost when another file is

deposited from a response from another receiver. When the administrator is prompted for the data base program names, the full pathnames must be specified. The executable code called 'im_code', the copy of the Shell controlling program called 'shell1', and the login id file called 'user.file' must be located in the current directory. See section 4.3.2 for a list of the programs and files necessary by process. The SENDER-INITIATE process requires write permission in the current directory.

When the Mail program unbundles the data object it creates several files, including the data object indicator, header file, executable code, data file, and Shell controlling program. The Mail program also creates several intermediary files. These files will be created in a subdirectory in the user's mail directory and removed when the execution of Mail is terminated. The following section describes the Mail program interaction in more detail.

4.6 *Mail Program Interaction*

When the Mail program scans the message it encounters certain standard Mail information, e.g., routing information, prior to the reading of the message itself. In the case of the intelligent message, the Mail program will encounter the data object indicator prior to the data object. This will trigger the unbundling of the data object and the execution of Shell commands. The Shell commands will execute the code which interacts with the receiver and stores updated information in the data file. If the receiver does not wish to complete the intelligent message, the Mail program will exit with the original message still intact, ready for rereading. If the receiver completes the intelligent message, the Mail program will rebundle the pieces of the data object, mail it back to the originator, and remove the original message. In either case, any data object files or intermediary files created as part of this process will be removed when the execution of Mail is terminated.

5. *CHAPTER FIVE - Implementation*

A partial implementation of 'XMAIL' has been completed. The implementation consists of four modified Mail 'C' programs and three modified Mail data files loaded into the Mail system, four Shell programs, one of which is conceptually part of the modified Mail program, and two data files. The remaining three Shell programs are test versions of the 'application code' (the data base interface programs and the executable code which were implemented seperately). The first data file is a login id file consisting of several login ids used for testing purposes. The second data file is a simulated data base retreival file. A copy of the Mail directory of programs and data files was made and these programs were modified to simulate the 'real' Mail environment. A test directory structure was created to simulate the /usr/spool/mail directory structure. See Appendix 2 for a copy of the code and data files. The Mail code shown is only the modified portion (shown thru a 'diff' of the real Mail code and the 'XMAIL' code).

All three processes (SENDER-INITIATE, RECEIVER, SENDER-RECEIVE RESPONSE) were tested. Data was simulated to be extracted from the database and intelligent messages were created and mailed (SENDER-

INITIATE process). Mail was invoked, the data object was processed, executable code was executed, responses were given, response messages created and mailed, and status information was sent to the sender (RECEIVER process). Mail was invoked, the data object was processed, and the updated data was used to simulate the update to the data base (SENDER-RECEIVE RESPONSE process). See Appendix 3 for the sample sessions. Testing was done to ensure that when a receiver does not complete updating through the intelligent message the message is saved for future processing. Testing was also done to ensure a clean-up of files.

6. *CHAPTER SIX - Conclusions and Extensions*

There were two problems encountered during the implementation of 'XMAIL' which made it a partial implementaion. One problem related to the automatic recognition of the intelligent data object by the Mail system. The solution was to create a new command, called 'go', to execute the message. The user/receiver will recognize an intelligent data object by the comments field printed with every message. The comment is "EXECUTABLE MESSAGE: use 'go' cmd". The help file was also changed to indicate the new command.

The second problem related to the automatic deletion of the message by the modified Mail system after the updating of the data was complete and the response was sent. The solution was to print a message telling the user/receiver to delete the message.

Also, the Shell controlling program required for 'XMAIL' should be located in a system directory and automatically picked up by 'XSEND'. The sender would then not need a copy in his/her directory.

'XMAIL' could be very useful in aiding the flow of information in an office environment. The system uses

the efficient and reliable electronic mail system rather than paper forms or questionnaires. The user/receiver sees little or no change in his/her environment and is not required to initiate a new system, since the user's invocation of the Mail system will be an invocation of 'XMAIL'. The updating of the data base is an important feature of the system. The entire process is automatic with no user intervention, once it is invoked.

'XMAIL' has many possible uses. The system could be used for completion of forms, for teacher evaluations, for surveys, etc. The application code (executable code and data base interface code) would have to be tailored to these needs.

A possible future enhancement is to allow forwarding or routing of the intelligent message, perhaps for completion or for evaluation. Another possible future enhancement is to allow the system to include user-defined messages, perhaps by utilizing an existing system.

Administrator Manual Page

NAME
  XSEND - initiate creating and sending of
          intelligent messages

SYNOPSIS
  XSEND [login_id]

DESCRIPTION
  The XSEND command initiates the creating and
  sending of intelligent messages.
  If there is a login_id given in the
  command line the intelligent message will
  be sent to that login id. If there is no
  login_id given in the command line a file
  of login ids named 'user.file' will be
  assumed to exist in the current directory.
  An intelligent message will be sent to
  each of the login ids in that file.

  Required files:
    1 - your data base retreival program that
        creates a data file named 'im_data'
        in the current directory (program
        has no naming or location restrictions)
    2 - your executable code to interface with
        the user to update the data file named
        'im_data' (code must be named 'im_code' and
        located in the current directory)
    3 - a copy of the Shell controlling program
        named 'shell1' (program must be
        located in the current directory)

Administrator Manual Page

NAME
Mail - send and recieve messages

SYNOPSIS
.
.
.

SUMMARY
.
.
.
go Execute an intelligent data object.

Original message:
If this is the first time this message is being read, status information will be sent back to the sender. The executable code supplied to the data object via the XSEND command will be executed. When data update is complete, the updated data will be sent back to the sender as part of an intelligent data object.
Response:
The updated data file will be moved to the directory of your choice under the name '${receiver}im_data'. Your data base update program (no location or naming restrictions) that accesses the data file '${receiver}im_data' will be executed if you wish.

Code

'Makefile' Differences

---------- ----------

(< original code, > XMAIL code)

```
7c7
< DESTDIR=
---
> DESTDIR= /usre/att/humphrey
43c43
<       $(RM) -f Mail
---
>       $(RM) -f XMAIL
45,46c45,48
<       @$(CC) -n -o Mail $(OBJS) $(LIBES)
<       @size Mail
---
>       @$(CC) -g -n -o XMAIL $(OBJS) $(LIBES)
>       @size XMAIL
>       cp XMAIL /usre/att/humphrey/bin/XMAIL
>       cp misc/Mail.help* /usre/att/humphrey/bin
52c54
<       install -s Mail ${DESTDIR}/usr/ucb/Mail
---
>       install -s XMAIL ${DESTDIR}/usr/ucb/XMAIL
55d56
<       cd ${DESTDIR}/usr/ucb; rm -f mail; ln Mail mail
70c71
<       rm -f Mail a.out fmt x.c xs.c tags core
---
>       rm -f XMAIL a.out fmt x.c xs.c tags core
```

Code

'cmd3.c' Differences

(< original code, > XMAIL code)

```
811a812,882
>
>
> /*
>  *
>  *
>  *    routine to execute a message
>  *    that is a data object
>  *
>  */
>
> go(msgvec)
>       int *msgvec;
> {
>       struct message *mailp;
>       FILE *exec_ar;
>       FILE *tfile;
>       char maildir[100];
>       char origdir[100];
>       char cmd[100];
>       char *origdirptr=origdir;
>       register *ip;
>       register int mesg;
>
>       mailp = &message[*msgvec - 1];
> /* touch message to indicate that message has been read and not
>    to be sent back to system mailbox on exit  */
>       ip = msgvec;
>       mesg = *ip;
>       touch(mesg);
>
>       system("pwd");
>
```

```
> /* save original directory to be able to return to it */
>       origdirptr = getwd(origdir);
>
> /* make new directory name for storage of temporary files */
>       copy(mailname,maildir);
>       stradd(maildir,'1');
>
> /* create the directory */
>       sprintf(cmd,"mkdir %s",maildir);
>       system(cmd);
>
> /* cd to the directory */
>       chdir(maildir);
>       system("pwd");
>
> /* strip the extra lines created by the mail system */
> /* to be able to un-archive the data object         */
>       exec_ar=popen("sed 1,/Status/d > arfile1","w");
>       send(mailp,exec_ar,0);
>       pclose(exec_ar);
>       system("sed 1d arfile1 > arfile2");
>       system("ar x arfile2");
>
> /* execute the data object pieces !!!!!! */
>       system("ls -l ");
>       system("cp shell1 shell2;chmod 755 shell2;sh shell2");
>       system("pwd");
>
> /* cd back to the original directory */
>       chdir(origdir);
>       system("pwd");
>
> /* remove the directory created for the data object */
>       sprintf(cmd,"rm -r %s",maildir);
>       system(cmd);
>
>
>       return(0);
> }
```

Code

'cmdtab.c' Differences

(< original code, > XMAIL code)

```
20c20
< extern int local(), folders(), igfield(), Type();
---
> extern int local(), folders(), igfield(), Type(), go();
81a82
>       "go",           go,     MSGLIST,        0,      MMNDEL,
```

Code

'lock.c' Differences

-------- ----------

(< original code, > XMAIL code)

```
15c15
< char  *lockname       = "/usr/spool/mail/tmXXXXXX";
---
> char  *lockname       = "/usre/att/humphrey/spool/mail/tmXXXXXX";
```

Code

'v7.local.c' Differences

---------- ----------

(< original code, > XMAIL code)

```
24c24
<       cp = copy("/usr/spool/mail/", mailname);
---
>       cp = copy("/usre/att/humphrey/spool/mail/", mailname);
```

Code

'v7.local.h' Differences

---------- ----------

(< original code, > XMAIL code)

```
16,17c16,17
< #define MAIL            "/bin/mail"     /* Name of mail sender */
< #define SENDMAIL        "/usr/lib/sendmail"
---
> #define MAIL            "/usre/att/humphrey/bin/XMAIL"/* Name of
                                                        mail sender */
> #define SENDMAIL        "/usre/att/humphrey/bin/XMAIL"
23c23
< #define HELPFILE        "/usr/lib/Mail.help"
---
> #define HELPFILE        "/usre/att/humphrey/usr/lib/Mail.help"
25c25
< #define THELPFILE       "/usr/lib/Mail.help. "
---
> #define THELPFILE       "/usre/att/humphrey/usr/lib/Mail.help. "
30c30
< #define MASTER          "/usr/lib/Mail.rc"
---
> #define MASTER          "/usre/att/humphrey/usr/lib/Mail.rc"
```

Code

'Mail.help'

```
    Mail   Commands
t <message list>         type messages
n                        goto and type next message
e <message list>         edit messages
f <message list>         give head lines of messages
d <message list>         delete messages
s <message list> file    append messages to file
u <message list>         undelete messages
r <message list>         reply to messages
pre <message list>       make messages go back to /usr/mail
m <user list>            mail to specific users
go <message list>        execute a data object message
q                        quit, saving unresolved messages in mbox
x                        quit, do not remove system mailbox
h                        print out active message headers
!                        shell escape
c [directory]            chdir to directory or home if none given

A <message list> consists of integers, ranges of same, or user names
separated by spaces.  If omitted, Mail uses the last message typed.

A <user list> consists of user names or distribution names separated
by spaces.
Distribution names are defined in .sendrc in your home directory.
```

Code

'db_get'

(data base retreival program)

```
# temporary code to simulate data base retrieval
#

echo retrieving data base information for ${1}

echo 999 > im_data
echo 23 >> im_data
echo 44 >> im_data
echo 109 >> im_data
echo 742 >> im_data

echo
echo data base information retrieved is as follows:
cat im_data
```

Code

'db_put'

(data base update program)

```
# temporary code to simulate data base update
# using the updated data in the im_data
#

echo Data received from user is as follows:
cat im_data
echo
echo Updating data base information.
```

Code

'im_code'

(executable code)

```
# temporary code to simulate interaction with user
#
echo interacting with user
echo 73 >> im_data

echo
echo modified data is as follows:
cat im_data
```

Code

'shell1'

(main controlling program)

```
# Shell commands to be included as part of the data object
#

cut -f1 -d" " header > header1
sender=`head -1 header1`
receiver=`head -2 header1 | tail -1`
status=`tail -1 header1`


# if mail is in first stage (mail sent),
# set status to second stage (mail received)
#   and send status info back to originator
#
if
        test "${status}" -eq "1"
then
        status=2
# until "status = 2" is saved in the mail message
# this question must be asked
        echo "Is this the first time you are "
        echo "reading this message? (y/n)"
        read ans
        if
          test "${ans}" = "y"
        then
          echo "Status information is being mailed to the sender."
          echo "User ${receiver} has received data object." >
                ${sender}status
          XMAIL -s "DATA OBJECT STATUS" ${sender} < ${sender}status
        fi
fi
```

```
# if mail is in second stage (mail received),
# execute the executable code
#
if
        test "${status}" -eq "2"
then

# program to interface with user and update data

        im_code
#
# if user is finished running the executable code
# set status to third stage and send response
# else leave files as is, to be completed later
#
# response will be sent from Mail Shell commands
#
        echo "Are you finished with your data entry? (y/n)"
        read ans


        if
          test "${ans}" = "y"
        then
          echo "Responses will be sent back to the originator."
          echo "Please delete this mail message."
          status=3
          echo ${sender} > header
          echo ${receiver} >> header
          echo ${status} >> header
          ar qc ${sender}obj indicator header im_code im_data shell1
          XMAIL -s "EXECUTABLE MSG: use 'go' cmd" ${sender}
                        < ${sender}obj
#       else need to save new "status = 2" in the mail message
        else
          echo "You may re-execute this message later."
          echo "You will need to re-enter all responses."
        fi
#
# if mail is not in second stage (mail received),
# then mail is in third stage (response sent) and
# execute data base interface program
#
else
        echo "This is a reply from ${receiver}."

# move data file to unique file relative to user
# to not lose file when next response is sent
#
        echo "Enter the directory where you want "
        echo "the data to be stored: "
```

```
        read dirnm
        cp im_data ${dirnm}/${receiver}im_data

        echo "Do you wish your data base to be updated now? (y/n)"
        read ans
        if test "${ans}" = "y"
        then
          echo "Enter the pathname of the local program "
          echo "to be executed to update the data base: "
          read filenm
          ${filenm}
        fi

fi
```

Code

'XSEND'

(main sending program)

```
# main controlling program for the INTELLIGENT MESSAGE SYSTEM
#
# either mail an intelligent message to one user
# using the first argument for the destination
# or mail intelligent messages to several users
# using a default file of login names for the destinations
#
#
# for each user do:
#
# - initiate data base interface program which will
#    retrieve information and put it into a file
#
# - create a data object containing:
#       data object indicator
#       header
#       executable code
#       data
#       Shell commands
#
# - mail the data object to the user
#

echo "*** DATA OBJECT ***" > indicator
who am i | awk '{ print $1 }' > mylogin
echo 1 > status
echo "Enter the pathname of the local program to be executed"
echo "to retrieve the data base information: "
read filenm

# if first argument is blank - send message to users
#   in the default file of login names
```

```
#
if
        test "${#}" -lt 1

then
        for user
        in `cat user.file`
        do

# program to retrieve data base information
# and put into a file
#
          echo Removing any existing data object.
          rm ${user}obj
          ${filenm} ${user}
          echo ${user} > receiver
          cat mylogin receiver status > header

# 'intelligent message code'
# program to interface with user and update data

# 'intelligent message data'
# data file containing user's data base information

          ar qc ${user}obj indicator header im_code im_data shell1

          XMAIL -s "EXECUTABLE MSG: use 'go' cmd" ${user}
                        < ${user}obj

        done

# if first argument is non-blank - send message to user
#       indicated in the argument
#
else

        echo Removing any existing data object.
        rm ${1}obj
        ${filenm} ${1}
        echo ${1} > receiver
        cat mylogin receiver status > header

        ar qc ${1}obj indicator header im_code im_data shell1

        XMAIL -s "EXECUTABLE MSG: use 'go' cmd" ${1}
                < ${1}obj
fi
```

Sample Sessions

SENDER-INITIATE Process

```
$ XSEND humphrey
Enter the pathname of the local program to be executed
to retreive the data base information:
/usre/att/humphrey/imp/tst3/db_get
Removing any existing data object.
retrieving data base information for humphrey
data base information retrieved is as follows:
999
23
44
109
742
```

Sample Sessions

RECEIVER Process

-------- -------

Initial Execution

```
$ XMAIL
Mail version 2.18 5/19/83.  Type ? for help.
"/usre/att/humphrey/spool/mail/humphrey": 1 message
>1 humphrey Mon Jul  6 11:07  130/3190 "EXECUTABLE MSG: use 'go' cmd"
& go 1
/usre/att/humphrey/imp/tst3
/usre/att/humphrey/spool/mail/humphrey1
total 11
-rw-------  1 humphrey     2954 Jul  6 11:20 arfile1
-rw-------  1 humphrey     2953 Jul  6 11:20 arfile2
-rw-------  1 humphrey       28 Jul  6 11:20 header
-rwx------  1 humphrey      178 Jul  6 11:20 im_code
-rw-------  1 humphrey       18 Jul  6 11:20 im_data
-rw-------  1 humphrey       20 Jul  6 11:20 indicator
-rwx------  1 humphrey     2400 Jul  6 11:20 shell1
Is this the first time you are
reading this message? (y/n)
y
Status information is being mailed to the sender.
interacting with user
modified data is as follows:
999
23
44
109
742
73
Are you finished with your data entry? (y/n)
n
You may re-execute this message later.
You will need to re-enter all responses.
/usre/att/humphrey/spool/mail/humphrey1
/usre/att/humphrey/imp/tst3
& q
Held 1 message in /usre/att/humphrey/spool/mail/humphrey
```

Sample Sessions

RECEIVER Process

Secondary Execution

```
$ XMAIL
Mail version 2.18 5/19/83.  Type ? for help.
"/usre/att/humphrey/spool/mail/humphrey": 1 message
>1 humphrey Mon Jul  6 11:07  130/3190 "EXECUTABLE MSG: use 'go' cmd"
& go  1
/usre/att/humphrey/imp/tst3
/usre/att/humphrey/spool/mail/humphrey1
total 11
-rw-------  1 humphrey     2954 Jul  6 11:25 arfile1
-rw-------  1 humphrey     2953 Jul  6 11:25 arfile2
-rw-------  1 humphrey       28 Jul  6 11:25 header
-rwx------  1 humphrey      178 Jul  6 11:25 im_code
-rw-------  1 humphrey       18 Jul  6 11:25 im_data
-rw-------  1 humphrey       20 Jul  6 11:25 indicator
-rwx------  1 humphrey     2400 Jul  6 11:25 shell1
Is this the first time you are
reading this message? (y/n)
n
interacting with user
modified data is as follows:
999
23
44
109
742
73
Are you finished with your data entry? (y/n)
y
Responses will be sent back to the originator.
Please delete this mail message.
/usre/att/humphrey/spool/mail/humphrey1
/usre/att/humphrey/imp/tst3
& q
Held 1 message in /usre/att/humphrey/spool/mail/humphrey
```

Sample Sessions

Status Information

```
$ XMAIL
Mail version 2.18 5/19/83.  Type ? for help.
"/usre/att/humphrey/spool/mail/humphrey": 3 messages 2 new 3 unread
 U1 humphrey Mon Jul  6 11:07  130/3189 "EXECUTABLE MSG: use 'go' cmd"
>N2 humphrey Mon Jul  6 11:20  9/265  "DATA OBJECT STATUS"
 N3 humphrey Mon Jul  6 11:25  131/3191 "EXECUTABLE MSG: use 'go' cmd"
& 2
Message  2:
From humphrey Mon Jul  6 11:20:48 1987
Date: Mon, 6 Jul 87 11:20:45 CDT
From: humphrey (ATT Cynthia Humphrey)
To: ksuvax1!humphrey
Subject: DATA OBJECT STATUS

User humphrey has received data object.

& q
Held 3 messages in /usre/att/humphrey/spool/mail/humphrey
```

Sample Sessions

SENDER-RECEIVE RESPONSE Process

```
$ XMAIL
Mail version 2.18 5/19/83.  Type ? for help.
"/usre/att/humphrey/spool/mail/humphrey": 3 messages 2 unread
>U1 humphrey Mon Jul  6 11:07  130/3189 "EXECUTABLE MSG: use 'go' cmd"
  2 humphrey Mon Jul  6 11:20  10/276 "DATA OBJECT STATUS"
 U3 humphrey Mon Jul  6 11:25  132/3201 "EXECUTABLE MSG: use 'go' cmd"
& go 3
/usre/att/humphrey/imp/tst3
/usre/att/humphrey/spool/mail/humphrey1
total 11
-rw-------  1 humphrey     2958 Jul  6 11:28 arfile1
-rw-------  1 humphrey     2957 Jul  6 11:28 arfile2
-rw-------  1 humphrey       28 Jul  6 11:28 header
-rwx------  1 humphrey      178 Jul  6 11:28 im_code
-rw-------  1 humphrey       21 Jul  6 11:28 im_data
-rw-------  1 humphrey       20 Jul  6 11:28 indicator
-rwx------  1 humphrey     2400 Jul  6 11:28 shell1
This is a reply from humphrey.
Enter the directory where you want the data to be stored:
/usre/att/humphrey/imp/tst3
Do you wish your data base to be updated now? (y/n)
y
Enter the pathname of the local program
to be executed to update the data base:
/usre/att/humphrey/imp/tst3/db_put
Data received from user is as follows:
999
23
44
109
742
73
Updating data base information.
/usre/att/humphrey/spool/mail/humphrey1
/usre/att/humphrey/imp/tst3
& q
Held 3 messages in /usre/att/humphrey/spool/mail/humphrey
```

[BERN82] Bernal, Marc, "Interface Concepts for Electronic Forms Design and Manipulation", Office Information Systems, N. Naffah(ed.), INRIA/North-Holland Pub. Co., 1982.

[COHE84] Cohen, A. Toni, "Data Abstraction, Data Encapsulation, and Object- Oriented Programming", SIGPLAN Notices, Vol. 19, No. 1, January 1984

[COX84] Cox, Brad J., "Message/Object Programming: An Evolutionary Change in Programming Technology", IEEE, January 1984

[ELLI80] Ellis, Clarence A. and Nutt, Gary J., "Office Information Systems and Computer Science", Computing Surveys, Vol. 12, No. 1, March 1980

[ELLI82] Ellis, Clarence A. and Bernal, Marc, "Officetalk-D: An Experimental Office Information System", ACM 0-89791-075-3/82/006/0131

[FIKE81] Fikes, Richard E., "Odyssey: A Knowledge Based Assistant", 0004-3702/81/0000-0000/North-Holland, 1981.

[GRIE77] Gries, David and Gehani, Narain, "Some Ideas on Data Types in High-Level Languages", Communications of the ACM, June 1977, Vol. 20, No. 6

[HOGG84] Hogg, John and Gamvroulas, Stelios, "An Active Mail System", Sigmod Record, Vol. 14, No. 2, 1984

[LISK74] Liskov, Barbara and Zilles, Stephen, "Programming With Abstract Data Types", SIGPLAN, April 1974

[MAZE83] Mazer, Murray S. and Lochovsky, Fredrick H., "Routing Specification In A Message Management System", Proceedings of the 16th Annual Hawaii Int'l Conf. on System

Sciences, 1983, Vol. 1

[MCBR83] McBride, R. A. and Unger, E. A., "Modeling Jobs In A Distributed System", 1983 ACM 0-89791-123-7/83/012/0032

[SORE79] Sorenson, P.G. and Wald, J.A. and Gustafson, J.C., "A Distributed Database Query System Based on a Forms Interface Processor", Proceedings of CIPS Session '79, Quebec City, June 1979.

[STEF81] Stefferud, E. and Mchugh, J., "The Role Of Computer Mail In Office Automation"

[TSIC82] Tsichritzis, D.C. and Rabitti, F.A. and Gibbs, S. and Nierstrasz, O.M. and Hogg, J., "A System For Managing Structured Messages", IEEE Trans. Commun., COM-30, 1(Jan 1982), pp. 66-73

[TSIC80] Tsichritzis, D., "OFS: An Integrated Form Management System", Proceedings Of The ACM International Conference On Very Large Data Bases, 1980

[UNIX84] UNIX Programmer's Manual, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California 94720, March 1984.

[VITT81] Vittal, John, "Active Message Processing: Messages As Messengers", North- Holland Publishing Company, IFIP, 1981

[YAO84] Yao, S. Bing, Hevner, Alan R., Shi, Zhongzhi, Luo, Dawei, "FORMANAGER: An Office Forms Management System", ACM Transactions on Office Information Systems, Vol. 2, No. 3, July 1984.

[ZISM78] Zisman, Michael D., "Office Automation: Revolution or Evolution?", Sloan Management Review, Spring 1978

An Intelligent Message System Implemented as a Data Object

by

Cynthia L. Humphrey

B. S. Millersville University, 1979

An Abstract of a Master's Report

submitted in partial fulfillment of the

requirements for the degree

Master of Science

Department of Computer Science

Kansas State University
Manhattan, Kansas

1987

An Intelligent Message System

Implemented as a Data Object

by Cynthia L. Humphrey

An Abstract of a Master's Report

'XMAIL', an intelligent mail message system implemented as an intelligent data object(IDO), is introduced in this paper. An intelligent message is an executable object with the ability to perform some actions and interact with the receiver. An IDO contains routing information and both data and program code to retrieve, manipulate and store the data. The system was implemented on a UNIX based system, utilizing a data base management system. The literature reviews, requirements, design, implementation and possible enhancements are discussed. A copy of the code is included in the appendices. Sample sessions of 'XMAIL' are also included in the appendices.

Although 'XMAIL' was implemented for a specific usage, the basic design can be tailored to other purposes by supplying different application code. The application code interfaces with the user and with the database. The system code is used by the system administrator to create the IDO, mail the object, receive and execute the object, and return the object.